Parametric Random Generation of Deterministic Tree Automata[☆,☆☆]

Pierre-Cyrille Héam[a,c], Cyril Nicaud[b], Sylvain Schmitz[c]

[a] *LIFC, Université de Franche-Comté & INRIA, Besançon, France*
[b] *LIGM, Université Paris Est & CNRS, Marne-la-Vallée, France*
[c] *LSV, ENS Cachan & CNRS & INRIA, Cachan, France*

Abstract

Uniform random generators deliver a simple empirical means to estimate the average complexity of an algorithm. We present a general rejection algorithm that generates sequential letter-to-letter transducers up to isomorphism. We also propose an original parametric random generation algorithm to produce sequential letter-to-letter transducers with a fixed number of transitions. We tailor this general scheme to randomly generate deterministic tree walking automata and deterministic top-down tree automata. We apply our implementation of the generator to the estimation of the average complexity of a deterministic tree walking automata to nondeterministic top-down tree automata construction we also implemented.

1. Introduction

The widespread use of automata as primitive bricks in computer science motivates an ever renewed search for efficient algorithms taking automata as input (see for some recent examples [1, 2, 3]). Developing new algorithms and heuristics raises crucial evaluation issues, as improved worst-case complexity upper-bounds do not always transcribe into clear practical gains [4].

A suite for software performance evaluation can usually gather three types of entries:[1]

1. benchmarks, i.e. large sets of typical samples, which can be prohibitively difficult to collect, and thus only exist for a few general problems,
2. hard instances, that provide good estimations of the worst case behaviour, but are not always relevant for average case evaluations,
3. random inputs, that deliver average complexity estimations, for which the catch resides in obtaining a meaningful random distribution (for instance a uniform random distribution). As the mathematical computation of the average complexity of an algorithm is an intricate

[☆] A preliminary version of this work was published under the title *Random Generation of Deterministic Tree (Walking) Automata* in Maneth, S., editor, *CIAA'09*, volume 5642 of *Lecture Notes in Computer Science*, pages 115–124. Springer, 2009. doi\string:10.1007/978-3-642-02979-0_15.

[☆☆] This work was supported in part by ANR projects GAMMA (BLAN07-2_195422), RAVAJ (SETIN-2006), and AVeriSS (SETIN-2006).

Email addresses: heampc@lifc.univ-fcomte.fr (Pierre-Cyrille Héam), cyril.nicaud@univ-mlv.fr (Cyril Nicaud), sylvain.schmitz@lsv.ens-cachan.fr (Sylvain Schmitz)

[1] All of the three types are used in SAT-solver competitions like http://www.satcompetition.org/.

task that cannot be undertaken in general, random inputs can prove themselves invaluable for its empirical estimation.

This paper is dedicated to the random generation of deterministic tree automata. Tree automata have witnessed a recent surge of interest in connection with XML applications [5, 6], fostering a wealth of theoretical results (e.g. [7, 8, 9]). This paper makes the following contributions:

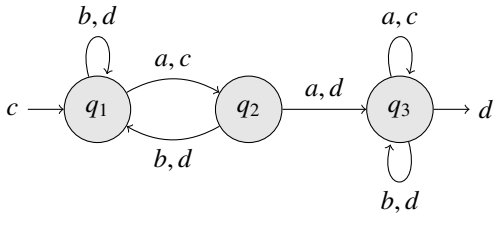
- Section 3 proposes a generic rejection algorithm for uniformly generating accessible sequential letter-to-letter transducers. Thanks to the structural properties of these transducers, the algorithm can be used for the generation of various kinds of finite automata.
- In Section 3.2 we propose an original parametric algorithm for uniformly generating accessible sequential letter-to-letter transducers with a fixed number of transitions. This generation technique is useful in order to evaluate how the performance of an algorithm depends on the density of transitions, which is a recurrent concern in empirical evaluations [10, 1, 11].
- We apply these algorithms in Section 4 to the generation of deterministic tree walking automata. The approach was implemented, and we provide in Section 4.3 an empirical estimation of the average size of the nondeterministic top-down tree automaton equivalent to a given deterministic tree walking automaton.
- Section 5 presents a bijection between a class of letter-to-letter transducers and deterministic top-down tree automata, providing a uniform random generator for this class of tree automata.
- We argue in Section 6 that our approach is suitable to randomly generate many other classes of finite state machines: we illustrate the cases of deterministic Turing machines, deterministic real-time pushdown automata, and deterministic visibly pushdown automata. The latter class of systems allows to circumvent the restrictions in Sections 4 and 5, as deterministic visibly pushdown automata can represent any regular tree language [12], and are especially suited for representing XML streams.

Our approach consists in reducing the problem to the uniform random generation of deterministic word automata, as developed by Bassino and Nicaud [13], Bassino et al. [14].

Related Work. In the case of deterministic accessible word automata, two main approaches to the random generation with uniform distribution on complete automata stand out: one based on a recursive decomposition [15] and one using Boltzmann samplers [13]. The latter algorithm has been extended to possibly incomplete automata by Bassino et al. [14]. An implementation of these algorithms is available in the C++ package `REGAL` [16].[2]

The problem of randomly generating nondeterministic finite word automata is still mostly open. Two recent papers propose such random generation algorithms: Tabakov and Vardi [10] apply theirs to the evaluation of inclusion testing procedures, whereas Chen et al. [17] evaluate the performance of a learning algorithm. Both algorithms are ad hoc, using ideas from random graph theory, and a theoretical analysis of the observed properties may be the next step for defining a class of meaningful distributions for NFAs.

[2] Available at `http://regal.univ-mlv.fr/`.



$\delta(q_1, a) = q_2$ $\quad \delta(q_2, a) = q_3$ $\quad \delta(q_3, a) = q_3$
$\delta(q_1, b) = q_1$ $\quad \delta(q_2, b) = q_1$ $\quad \delta(q_3, b) = q_3$
$\gamma(q_1, a) = c$ $\quad \gamma(q_2, a) = d$ $\quad \gamma(q_3, a) = c$
$\gamma(q_1, b) = d$ $\quad \gamma(q_2, b) = d$ $\quad \gamma(q_3, b) = d$
$\rho(q_3) = d$

$a_{\text{init}} = c$

Figure 1: A sequential letter-to-letter transducer.

2. Preliminaries

If i and j are positive integers, we denote by $[i, j]$ the set of integers k such that $i \leq k$ and $k \leq j$. If K is a set, $\mathcal{P}(K)$ (resp. $\mathcal{P}^*(K)$) denotes the set of subsets (resp. the set of nonempty subsets) of K. The domain of a function φ is denoted $\text{Dom}(\varphi)$.

Sequential Transducers. A *sequential letter-to-letter transducer* (SLT) from input alphabet Σ_1 to output alphabet Σ_2 is a tuple $\mathcal{T} = (\Sigma_1, \Sigma_2, Q, q_{\text{init}}, \delta, \gamma, \rho, a_{\text{init}})$ where Q is the finite set of *states*, $q_{\text{init}} \in Q$ is the *initial state*, δ is a partial *transition function* from $Q \times \Sigma_1$ into Q, γ is a partial *output function* from $Q \times \Sigma_1$ into Σ_2 such that $\text{Dom}(\delta) = \text{Dom}(\gamma)$, ρ is a partial *final function* from Q into Σ_2, and $a_{\text{init}} \in \Sigma_2$ is the *initial output*. An SLT is *complete* if $\text{Dom}(\delta) = Q \times \Sigma_1$. *Accessible* states of an SLT are inductively defined by: q_{init} is accessible and if q is accessible, for every $a \in \Sigma_1$, $\delta(q, a)$ is accessible. An SLT is *accessible* if all its states are accessible. An example of a complete and accessible SLT is depicted in Figure 1.

Let $\mathcal{T}_1 = (\Sigma_1, \Sigma_2, Q_1, q_{\text{init1}}, \delta_1, \gamma_1, \rho_1, a_{\text{init1}})$ and $\mathcal{T}_2 = (\Sigma_1, \Sigma_2, Q_2, q_{\text{init2}}, \delta_2, \gamma_2, \rho_2, a_{\text{init2}})$ be two SLTs. A function φ from Q_1 to Q_2 is an *isomorphism* from $\mathcal{T}_1$ to $\mathcal{T}_2$ if it satisfies the following conditions:

1. φ is bijective,
2. $\varphi(q_{\text{init1}}) = q_{\text{init2}}$,
3. $\delta_1(q, a) = p$ iff $\delta_2(\varphi(q), a) = \varphi(p)$,
4. $\gamma_1(q, a) = b$ iff $\gamma_2(\varphi(q), a) = b$,
5. $\rho_1(q) = b$ iff $\rho_2(\varphi(q)) = b$, and
6. $\varphi(a_{\text{init1}}) = a_{\text{init2}}$.

If such an isomorphism exists, we say that $\mathcal{T}_1$ and $\mathcal{T}_2$ are isomorphic. Informally, $\mathcal{T}_1$ and $\mathcal{T}_2$ are isomorphic if they encode the same SLT, up to state names. The relation *is isomorphic to* is trivially an equivalence relation. More formal machines will be introduced as needed later.

In this paper, we are interested in the uniform random generation of SLTs up to isomorphism, i.e. we want to equiprobably generate equivalence classes for the isomorphic relation (and for a given number of states). Since the approach is purely syntactic and will be applied to different classes of finite automata, the semantics of SLTs is moslty irrelevant; its only interest is to justify our focus on *accessible* states.

Algorithm 1: A rejection algorithm for $\mathsf{Generate}_X()$

```
1 repeat
2 |  e ← Generate_Y()
3 until e ∈ X
4 return e
```

Rejection Algorithms. Before we describe our generation algorithms, let us recall the definition of a *rejection algorithm*: Suppose we want to generate elements of a set X, according to a probability distribution p_X. Furthermore, suppose that X is a subset of Y, and that we have a probability distribution p_Y on Y, whose restriction to X is p_X. If we have an algorithm $\mathsf{Generate}_Y$ that generates elements of Y according to p_Y, we may use this algorithm to generate elements of X as follows: repeatedly draw an element of Y, reject it if it is not in X, and stop if it is in X (see Algorithm 1). Note that if p_Y is the uniform distribution on Y, then p_X is the uniform distribution on X.

The average complexity of this rejection algorithm depends on the complexity of the generation algorithm on Y (line 2), added with the complexity to test whether an element of Y is in X (line 3), and multiplied by the average number of iterations. One can see that if $p_Y(X)$ is the probability for an element of Y to be in X, the average number of iterations is $1/p_Y(X)$.

One could exploit directly the uniform random generator of Bassino et al. [14], by defining a bijection between the family of desired (tree) automata $\mathcal{T}_n$ and a subfamily X of the deterministic word automata $\mathcal{A}_n$, and by employing a rejection algorithm. We rather introduce in the next section a generic, intermediate step, based on *families of SLTs*, which allows us to give general complexity results for our generators.

3. Generating Sequential Transducers

We propose in this section general methods to generate randomly and uniformly deterministic and accessible automata-like structures with n states. To this end, we develop an algorithm that generates sequential letter-to-letter accessible transducers with n states, that can be further parametrized by giving

- some *restrictions* on the possible outputs for each input letter (Section 3.1),
- a number m of *missing* transitions (Section 3.2).

The idea thereafter, for each given problem, is to find an effective *bijection* φ between the structures one wants to generate and such a family of transducers.

The algorithm is in fact more general, since by Proposition 1, one can build an effective random generator even if φ is only an injection, provided that all the *complete* transducers are in the image of φ. This method will be applied in the following sections to build random generators for deterministic tree walking automata, deterministic top-down tree automata, and other families of deterministic automata.

Note that we are only interested here

- in the *combinatorial structures* of transducers, not on what their models are. Indeed, our approach will be used in order to generate several kinds of finite automata;

- in the uniform random generation of *isomorphic classes* of SLTs. The algorithms proposed in this section fulfill this criterion. However, in order to simplify the exposition, we will write about random generation of SLTs rather than of equivalence classes of SLTs, but keep in mind that we randomly generate witnesses of equivalence classes.

3.1. Generation with Output Restrictions

The idea to generate deterministic and accessible word automata developed by Bassino et al. [13, 14] is to exhibit an effective injection ι from automata with n states on a k-letter alphabet to partitions of $[1, kn + 1]$ in n parts in the complete case and of $[1, kn + 2]$ in $n + 1$ parts in the possibly incomplete case. The inverse ι^{-1} can also be computed, and though all partitions are not the image of an automaton, there are enough of them to guarantee that a rejection algorithm is efficient. The algorithm therefore consists in randomly generating a partition, using a Boltzmann sampler, until the partition is the image of an automaton, and then compute its preimage. Its average complexity is in $O(n^{3/2})$.

Families of Transducers. Let us consider the family $\mathcal{D}_n(\Sigma_1, \Sigma_2, r, r_i, r_F)$ of accessible SLTs with n states, where Σ_1 is the input alphabet, Σ_2 is the output alphabet, $r : \Sigma_1 \to \mathcal{P}^*(\Sigma_2)$ is a restriction on transitions, $r_i \in \mathcal{P}^*(\Sigma_2)$ is a restriction on initialization and $r_F \in \mathcal{P}^*(\Sigma_2)$ is a restriction on finalizations. An n-state accessible SLT $(\Sigma_1, \Sigma_2, Q, i, \delta, \gamma, \rho, a_i)$ belongs to $\mathcal{D}_n(\Sigma_1, \Sigma_2, r, r_i, r_F)$ if the following conditions are met:

1. $a_i \in r_i$,
2. $\rho(Q) \subseteq r_F$, and
3. for all $a \in \Sigma_1$, $\gamma(Q, a) \subseteq r(a)$.

We denote by $C_n(\Sigma_1, \Sigma_2, r, r_i, r_F)$ the subset of $\mathcal{D}_n(\Sigma_1, \Sigma_2, r, r_i, r_F)$ that contains all the complete transducers. In order to generate a random SLT of $\mathcal{D}_n(\Sigma_1, \Sigma_2, r, r_i, r_F)$ or $C_n(\Sigma_1, \Sigma_2, r, r_i, r_F)$, we split the problem into three parts: the underlying graph with input symbols, the transitions outputs, and the set of final states. For complete transducers, one can perform these parts independently and still ensure equiprobability. A rejection algorithm is used to adapt this method to possibly incomplete ones.

Complete SLTs. The algorithm to generate a random complete SLT of $C_n(\Sigma_1, \Sigma_2, r, r_i, r_F)$ consists of the following three steps:

1. Randomly generate a complete deterministic and accessible automaton on Σ_1.
2. For each $q \in Q$ and each $a \in \Sigma_1$, randomly and uniformly choose $\gamma(q, a)$ in $r(a)$.
3. For each $q \in Q$, randomly and uniformly choose an element x of $r_F \uplus \{\#\}$, where # is a new symbol indicating that the state is not final; then define $\rho(q) = x$ if $x \neq \#$ and leave $\rho(q)$ otherwise undefined.

One can give the number of final states as a parameter f and change Step 3 into: Choose a random subset F with f elements of Q, and for each $q \in F$, choose $\rho(q)$ in r_F. The average complexity of the algorithm remains in $O(n^{3/2})$.

Possibly Incomplete SLTs. In order to generate a random possibly incomplete SLT of the full $\mathcal{D}_n(\Sigma_1, \Sigma_2, r, r_i, r_F)$ family, we proceed as before, except that we generate a possibly incomplete automaton at Step 1. The problem here is that the distribution is not uniform anymore, since we consider multiple choices of $\gamma(q, a)$ when the transition does not exist, leading to the same transducer. In order to obtain uniformity, we arbitrarily order Σ_2 and only keep, using a rejection algorithm, transducers such that $\gamma(q, a)$ is set to the minimum in $r(a)$ for every undefined transition. Corollary 1 of [14] ensures that a proportion greater than c, where $c > 0$ is a real number, of possibly incomplete automata are complete. The average number of rejects of this method is therefore in $O(1)$, as complete structures are not rejected and are numerous enough. The average complexity is in $O(n^{3/2})$ as well. Observe that if we had generated the image of $\gamma(q, a)$ for defined transitions only, we would have lost uniformity.

Complexity. Using the same argument about the proportion of complete automata given in Corollary 1 of [14], we can prove the following fairly general proposition:

Proposition 1. *Let E_n be a subset of $\mathcal{D}_n(\Sigma_1, \Sigma_2, r, r_i, r_F)$ such that E_n contains $C_n(\Sigma_1, \Sigma_2, r, r_i, r_F)$. The rejection algorithm consisting in generating uniformly an element of $\mathcal{D}_n(\Sigma_1, \Sigma_2, r, r_i, r_F)$ until it is in E_n performs $O(1)$ iterations on average.*

Therefore, we have a straightforward method to build a random generator for such a class E_n, which is efficient if one can quickly test if a given transducer is in E_n. In particular, if the membership test can be done in linear time—which will be the case in all the following instances of this generation scheme—then the average complexity of this method is in $O(n^{3/2})$. Note that the constant factor might grow quickly, e.g. when $|\Sigma_2|$ grows.

3.2. Random Generation with a Fixed Number of Undefined Transitions

The previous algorithm for random generation of possibly incomplete structures tends to generate automata that are nearly complete. This section introduces a different technique that takes as parameter the number m of *missing* transitions compared to the complete automaton with n states. Although its computational complexity is higher, this technique allows to tweak very finely the shape of the generated automata.

For this section, $k = |\Sigma_1|$ denotes the size of the input alphabet, which is arbitrarily ordered by $a_1 < a_2 < \cdots < a_k$. We are interested in generating uniformly and randomly elements of $\mathcal{X}_n(\Sigma_1, \Sigma_2, r, r_i, r_f, m)$, which we define as the set of elements of $\mathcal{D}_n(\Sigma_1, \Sigma_2, r, r_i, r_F)$ with exactly $m \in \mathbb{N}$ undefined transitions, i.e. such that $|\mathrm{Dom}(\delta)| = kn - m$. In the sequel, the alphabets and the restriction functions are fixed, and we denote $\mathcal{X}_n(\Sigma_1, \Sigma_2, r, r_i, r_F, m)$ by $\mathcal{X}_n(m)$. The *underlying graph* of $\mathcal{A} \in \mathcal{X}_n(m)$ is the labeled graph obtained after removing the information about the output and final functions and about the initial output from $\mathcal{A}$.

As we are working up to isomorphism, we consider in this section that $Q = [1, n]$, that the initial state is 1 and that the states of an element of $\mathcal{X}_n(m)$ are labeled in breadth-first order. Under these conditions, two different elements of $\mathcal{X}_n(m)$ cannot be isomorphic, thus simplifying the enumerations.

First note that if $kn - m < n - 1$, i.e. $m > (k-1)n + 1$, then there are not enough transitions for the transducer to be accessible, and therefore $\mathcal{X}_n(m) = \emptyset$. We therefore assume in the following that

$$m \leq (k-1)n + 1 \ .$$

Also note that Proposition 1 does not apply for $m \neq 0$, as complete structures are not included in $\mathcal{X}_n(m)$.

Characterizing Accessible Underlying Graphs. To generate uniformly at random an element of $\mathcal{X}_n(m)$, we use a recursive method similar to that of Champarnaud and Paranthoën [15], but applied to the representation of possibly incomplete deterministic automata described by Bassino et al. [14]. Let $\mathcal{A}$ be in $\mathcal{X}_n(m)$. We order $Q \times \Sigma_1$ lexicographically, and denote by ν the unique non-decreasing mapping from $Q \times \Sigma_1$ onto $[1, kn]$. Therefore, for (q, a) and (q', a') in $Q \times \Sigma_1$, $\nu((q, a)) < \nu((q', a'))$ if and only if the transition labeled by a from q is considered before the one labeled by a' from q' when performing a breadth-first search from the initial state. The underlying graph of an element of $\mathcal{X}_n(m)$ can therefore be seen as a partial function $g_{\mathcal{A}}$ from $[1, kn]$ to $[1, n]$, with $g_{\mathcal{A}} = \delta \circ \nu^{-1}$. For every $i \in [1, kn]$ we denote by $q(i)$ and $a(i)$ the state and the letter such that $\nu((q(i), a(i))) = i$. Note that $q(i)$ and $a(i)$ can be computed using

$$\begin{cases} q(i) = ((i-1) \operatorname{div} k) + 1 \ , \\ a(i) = a_{((i-1) \bmod k)+1} \ . \end{cases} \tag{1}$$

The size of the domain of $g_{\mathcal{A}}$ is $kn - m$, and one can easily build the underlying graph of an element $\mathcal{A} \in \mathcal{X}_n(m)$ if $g_{\mathcal{A}}$ is given.

A partial function g from $[1, kn]$ to $[1, n]$, with $|\mathrm{Dom}(g)| = kn - m$, is however not always the function $g_{\mathcal{A}}$ of an element of $\mathcal{X}_n(m)$, because of the accessibility condition:

Lemma 1. *Let g be a partial function from $[1, kn]$ to $[1, n]$ such that $|Dom(g)| = kn - m$. There exists $\mathcal{A} \in \mathcal{X}_n(m)$ such that $g = g_{\mathcal{A}}$ if and only if for every $q \in [2, n]$, there exists $i \in [1, (q-1)k]$ such that $g(i) = q$.*

Proof. As ν enumerates the (possibly undefined) transitions in breadth-first order, the images of the elements of $[1, (q-1)\,k]$ exactly correspond via ν^{-1} to the states accessible from the states $[1, q-1]$ using one transition. And if a state q is accessible, it is accessible from a smaller state in breadth-first order. □

We denote by $\mathcal{F}_{n,m}$ the set of partial functions from $[1, kn]$ to $[1, m]$ satisfying the conditions of Lemma 1.

This construction is uniquely defined on underlying graphs [14, Theorem 2], therefore the number of distinct underlying graphs of elements of $\mathcal{X}_n(m)$ is exactly the number of elements of $\mathcal{F}_{n,m}$.

Enumerating SLTs. We are now interested in finding a formula for $|\mathcal{X}_n(m)|$. For a given $g \in \mathcal{F}_{n,m}$, corresponding to a unique underlying graph, one can obtain different elements of $\mathcal{X}_n(m)$ by choosing the output (with restriction r) for every defined transition, the initial output (with restriction r_i), and the set of final states and their outputs (with restriction r_F). Hence, the number of elements of $\mathcal{X}_n(m)$ having an underlying graph corresponding to g is

$$|r_i| \prod_{a \in \Sigma_1} |r(a)|^{n(g,a)} \left(\sum_{F \subseteq [1,n]} |r_F|^{|F|} \right) = |r_i|(1 + |r_F|)^n \prod_{a \in \Sigma_1} |r(a)|^{n(g,a)}$$

where $n(g, a)$ is the number of transitions labeled by a that are defined in the underlying graph associated to g. The number of elements of $\mathcal{X}_n(m)$ is therefore

$$|\mathcal{X}_n(m)| = |r_i|(1 + |r_F|)^n \sum_{g \in \mathcal{F}_{n,m}} \prod_{a \in \Sigma_1} |r(a)|^{n(g,a)} \ .$$

Note that the terms before the sum on the elements of $\mathcal{F}_{n,m}$ correspond to the choice of the initial output and to the choice of the final states and of their outputs. This can be done independently from the transitions and can also easily be parametrized: one can for instance fix the number of final states just like we fix the number of undefined transitions.

Transitions and their Outputs. We now focus on generating uniformly at random the transitions and their outputs.

For $N \in [1, kn-1]$, let $\mathcal{F}_{N,n,m}$ be the set of all partial functions t from $[1, N]$ to $[1, n] \times \Sigma_2$ such that:

1. $|\mathrm{Dom}(t)| = N - m$,
2. on the first coordinate, the condition of Lemma 1 is satisfied: for every $q \in [2, n]$, there exist $i \in [1, \min(N, (q-1)k)]$ and $\ell \in \Sigma_2$ such that $t(i) = (q, \ell)$, and
3. the outputs satisfy r: for every $i \in [1, N]$, if $t(i) = (q, \ell)$ is defined, then $\ell \in r(a(i))$, where $a(i)$ is the letter corresponding to transition i, defined in Equation (1).

Informally, we are considering the N first transitions only, when they already fulfill the accessibility condition. The idea is to build an inductive formula for $|\mathcal{F}_{N,n,m}|$, by removing the transitions one by one, from N to 1; this formula will directly give an algorithm to generate uniformly at random the underlying graph with output.

First remark that if $m = 0$, then the transition $\nu^{-1}(kn) = (n, a_k)$ in an element of $\mathcal{X}_n(0)$ is always defined, and can be any element of $[1, n]$ as the graph is already accessible before examining the transitions from state n. And if $m \geq 1$, the last transition can be defined or not, so that we can count the number $x_n(m)$ of underlying graphs with output by:

$$\begin{aligned} x_n(0) &= |r(a_k)| \cdot n \cdot |\mathcal{F}_{nk-1,n,0}| \,, \\ x_n(m) &= |r(a_k)| \cdot n \cdot |\mathcal{F}_{nk-1,n,m}| + |\mathcal{F}_{nk-1,n,m-1}| \qquad \text{for } m \geq 1 \,. \end{aligned}$$

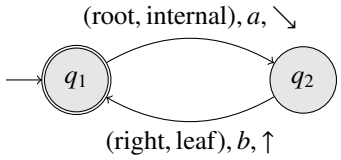
Let $f(N, n, m) = |\mathcal{F}_{N,n,m}|$ when $N < kn$ and $m \geq 0$, and $f(N, n, m) = 0$ otherwise. For an element t in $\mathcal{F}_{N,n,m}$ the N-th transition, when defined, can be in one of the following cases:

- It is the only transition reaching state n, i.e. $\exists a \in \Sigma_2, t(i) = (n, a) \Leftrightarrow i = N$. Hence if it is removed, the remaining accessible part is an element of $\mathcal{F}_{N-1,n-1,m}$. Note that from the accessibility condition, we must have $N - 1 < k\,(n-1)$ in this case, so that we can use this case again for a graph with $n-1$ states recursively.
- There is a smaller index $i \in [1, N-1]$ and some $a \in \Sigma_2$ such that $t(i) = (n, a)$, hence $t(N)$ can be any (q, a) in $[1, n] \times \Sigma_2$ satisfying the restriction condition on $a(N)$, or undefined if $m > 0$.

Putting all together, with initial conditions, we obtain:

$$\begin{cases} f(N,n,m) & = 0 \text{ if } N \geq kn \text{ or } m < 0 \text{ or } n \leq 0 \,, \\ f(N,n,m) & = 0 \text{ if } N - m < n - 1 \,, \\ f(0,1,0) & = 1 \,, \\ f(N,n,m) & = |r(a(N))|\, f(N-1, n-1, m) + |r(a(N))|\, n\, f(N-1, n, m) \\ & \quad + f(N-1, n, m-1) \quad \text{otherwise.} \end{cases} \tag{2}$$

The second condition ensures that there are enough transitions to reach every state.



$$\Delta(q_1, (\text{root}, \text{internal}), a) = (\searrow, q_2)$$
$$\Delta(q_2, (\text{right}, \text{leaf}), b) = (\uparrow, q_1)$$

Figure 2: A deterministic tree walking automaton.

At this point, random generation becomes straightforward: compute all the required values of $f(N, n, m)$, and from Equations (2) compute for each transition the probability that it is defined or not, and if it is, for any $(q, a) \in Q \times \Sigma_2$, the probability that it ends in q with output a. See Algorithm 2 for the details of the procedure; we assume that $\mathsf{Uniform}(X)$, where X is a finite set of elements, returns an element of X uniformly at random.

Complexity. Computing the values of $f(N', n', m')$ is the most expensive part, as stated in the following proposition.

Proposition 2. *Under the RAM model, the complexity of the precomputation step is $\Theta(n^3)$, both in time and space. After this precomputation, generating an element of $\mathcal{X}_n(\Sigma_1, \Sigma_2, r, r_i, r_f, m)$ can be performed in linear time.*

Note that the values reached by $f(N', n', m')$ can be huge, but that this kind of algorithms behaves well when using floating point approximations, giving only a small bias in uniformity. See Denise and Zimmermann [18] for more details on this point.

4. Application to Tree Walking Automata

4.1. Deterministic Tree Walking Automata

A *deterministic tree walking automaton* (DTWA) on binary trees is a tuple $(Q, \Sigma, \Delta, q_{\text{init}}, F)$ where Q is a finite set of states, $q_{\text{init}} \in Q$ is the initial state, $F \subseteq Q$ the set of final states and Δ is a partial transition function from $Q \times \text{TYPE} \times \Sigma$ to $\{\varepsilon, \uparrow, \swarrow, \searrow\} \times Q$, where $\text{TYPE} = \{\text{root}, \text{left}, \text{right}\} \times \{\text{internal}, \text{leaf}\}$. A deterministic tree walking automaton is *complete* if Δ is a complete function. Accessible states of a DTWA are defined inductively: q_{init} is accessible, and if q is accessible and $\Delta(q, t, a) = (d, p)$ for some $(t, a) \in \text{TYPE} \times \Sigma$, then p is accessible. An example of a DTWA is shown in Figure 2.

An *isomorphism* from a DTWA $(Q_1, \Sigma, \Delta_1, q_{\text{init}1}, F_1)$ to a DTWA $(Q_2, \Sigma, \Delta_2, q_{\text{init}2}, F_2)$ is a bijective function from Q_1 to Q_2 satisfying the three conditions (1) $\varphi(q_{\text{init}1}) = q_{\text{init}2}$, (2) $\varphi(q) \in F_2$ iff $q \in F_1$, and (3) $\Delta_1(q, t, a) = (d, p)$ iff $\Delta_2(\varphi(q), t, a) = (d, \varphi(p))$.

4.2. From SLTs to DTWAs

We define in this section a rather straightforward bijection τ between DTWAs and a class of SLTs, called *DTWA-coherent* SLTs, that contains all the complete SLTs. We obtain thereafter a random generation algorithm for DTWAs thanks to the restriction mechanisms introduced in Section 3.

Algorithm 2: Generate uniformly at random a SLT in $\mathcal{X}_n(m)$.

```
// Precomputation
Compute every f(N', n', m') for N' < kn, n' ≤ n and m' ≤ m
// The last transition
if m = 0 then
    δ(n, a_k) ← Uniform([1, n])
    γ(n, a_k) ← Uniform(r(a_k))
end
else
    if Uniform([1, x_n(m)]) ≤ f(nk − 1, n, m − 1) then
        δ(n, a_k) is undefined
        m ← m − 1
    end
    else
        δ(n, a_k) ← Uniform([1, n])
        γ(n, a_k) ← Uniform(r(a_k))
    end
end
N ← kn − 1
// The main loop
while N > 0 do
    dice ← Uniform([1, f(N, n, m)])
    if dice ≤ f(N − 1, n, m − 1) then
        δ(q(N), a(N)) is undefined
        m ← m − 1
    end
    else if dice − f(N − 1, n, m − 1) ≤ |r(a(N))| f(N − 1, n − 1, m) then
        δ(q(N), a(N)) ← n
        γ(q(N), a(N)) ← Uniform(r(a(N)))
        n ← n − 1
    end
    else
        δ(q(N), a(N)) ← Uniform([1, n])
        γ(q(N), a(N)) ← Uniform(r(a(N)))
    end
    N ← N − 1
end
Generate uniformly the final states and their outputs.
Generate uniformly the initial output.
return the transducer.
```

We first observe that a tree walking automaton can be viewed as an SLT with input alphabet Σ_1 and output alphabet Σ_2 defined by $\Sigma_1 = \text{TYPE} \times \Sigma$ and $\Sigma_2 = \{\varepsilon, \uparrow, \swarrow, \searrow\}$. Let

$\mathcal{A} = (Q, \Sigma, \Delta, q_{\text{init}}, F)$ be a DTWA; we define the SLT $\tau(\mathcal{A})$ by

$$\tau(\mathcal{A}) = (\Sigma_1, \Sigma_2 \uplus \{\$, 1\}, Q, q_{\text{init}}, \delta, \gamma, \rho, \$) ,$$

with $\delta(q, (t, a)) = p$ and $\gamma(q, (t, a)) = d$ iff $\Delta(q, t, a) = (d, p)$, and $\text{Dom}(\rho) = F$ with $\rho(q) = 1$ iff $q \in F$. For the example depicted in Figure 2,

$$\begin{array}{lll} \delta(q_1, ((\text{root}, \text{internal}), a)) = q_2 & \gamma(q_1, ((\text{root}, \text{internal}), a)) = \searrow & \\ \delta(q_2, ((\text{right}, \text{leaf}), b) = q_1 & \gamma(q_2, ((\text{right}, \text{leaf}), b) = \uparrow & \rho(q_1) = 1 \ . \end{array}$$

An SLT on $\Sigma_1, \Sigma_2 \uplus \{\$, 1\}$ is *DTWA-coherent* if its initial output symbol is \$ and if for every $q \in \text{Dom}(\rho)$, $\rho(q) = 1$.

Let us now provide an algorithm for random generation up to isomorphism of DTWAs. We re-use for this purpose the SLT generation algorithm, and need the following two propositions.

Proposition 3. *The function τ is a bijection from DTWAs to DTWA-coherent SLTs. Moreover, for every DTWA $\mathcal{A}$, $\tau(\mathcal{A})$ is complete (resp. accessible) if and only if $\mathcal{A}$ is complete (resp. accessible).*

Proof. Let $\mathcal{A}_1 = (Q_1, \Sigma, \Delta_1, q_{\text{init1}}, F_1)$ and $\mathcal{A}_2 = (Q_2, \Sigma, \Delta_2, q_{\text{init2}}, F_2)$ such that $\tau(\mathcal{A}_1) = \tau(\mathcal{A}_2) = (\Sigma_1, \Sigma_2 \uplus \{\$, 1\}, Q, q_{\text{init}}, \delta, \gamma, \rho, \$)$. By construction, $Q = Q_1 = Q_2$, $q_{\text{init}} = q_{\text{init1}} = q_{\text{init2}}$ and $\text{Dom}(\rho) = F_1 = F_2$. Moreover, $\delta(q, (t, a)) = p$ and $\gamma(q, (t, a)) = d$ iff $\Delta_1(q, t, a) = (d, p)$ iff $\Delta_2(q, t, a) = (d, p)$. Consequently $\Delta_1 = \Delta_2$, proving that τ is injective. Surjectivity of τ is a direct consequence of its definition. □

Proposition 4. *Two DTWAs $\mathcal{A}_1$ and $\mathcal{A}_2$ are isomorphic if and only if $\tau(\mathcal{A}_1)$ and $\tau(\mathcal{A}_2)$ are isomorphic.*

Proof. It suffices to note that the same isomorphism holds between $\mathcal{A}_1$ and $\mathcal{A}_2$ and $\tau(\mathcal{A}_1)$ and $\tau(\mathcal{A}_2)$. □

Restrictions on Output Functions. Moreover, the restrictions introduced in Section 3 are helpful in order to generate tree walking automata. Indeed, in a tree walking automaton, a transition labeled by $((t, a), d)$, with $(t, a) \in \Sigma_1$ and $d \in \Sigma_2$ is *useless* (i.e. can never be fired) in either of the following two cases:

1. t is in $\{\text{root}\} \times \{\text{internal}, \text{leaf}\}$ and $d = \uparrow$, or
2. t is in $\{\text{root}, \text{left}, \text{right}\} \times \{\text{leaf}\}$ and $d \in \{\swarrow, \searrow\}$.

Let us denote by r^{DTWA} the subset of $\Sigma_1 \times \Sigma_2$ of the pairs (a, b) that do not match any of the above two cases. The class E_n^{DTWA} of useful DTWA-coherent SLTs with n states then contains $C_n(\Sigma_1, \Sigma_2 \uplus \{\$, 1\}, r^{\text{DTWA}}, \{\$\}, \{1\})$ and is included in $\mathcal{D}_n(\Sigma_1, \Sigma_2 \uplus \{\$, 1\}, r^{\text{DTWA}}, \{\$\}, \{1\})$. Thus, random generation of DTWAs can be performed by first using Proposition 1 or Proposition 2 to obtain a SLT $\mathcal{T}$ and then by computing $\tau^{-1}(\mathcal{T})$.

A Normal Form for DTWAs. Tree-walking automata are especially useful as a means to define relations between nodes of a tree; however, when seen as tree language acceptors, there is little point in allowing several final states or outgoing transitions from final states. Uniform random generation of n-states DTWAs of this particular form does not fit our framework as such, and requires specific handling. We point to a sensible solution for generating such automata, but leave the details for future work.

Formally, a DTWA $(Q, \Sigma, \Delta, q_{\text{init}}, F)$ is in *final normal form* if F is a singleton $\{q_f\}$ and $\Delta(q_f, t, a)$ is undefined for all t in TYPE and a in Σ. It is in *complete final normal form* if the restriction of Δ to $(Q \backslash \{q_f\}) \times \text{TYPES} \times \Sigma$ is a total function.

Generating n-states DTWAs in complete final normal form uniformly at random can be performed by a rejection algorithm: generate a $(n-1)$-states possibly incomplete DTWA uniformly at random using our framework, but reject if the DTWA is complete. Given such an incomplete $(n-1)$-states automaton, one obtains an n-states DTWA in complete final normal form by (1) forgetting the final states information, (2) adding a new single final state q_f, and (3) having all the missing transitions point to q_f and choosing uniformly at random their directions in Σ_2. Assuming that a non negligible proportion of possibly incomplete automata are incomplete—which is seconded by the experimental results in Section 6.3 of Bassino et al. [14], where more than 80% of the possibly incomplete automata on alphabets of size larger than 2 are incomplete—then the average complexity remains in $O(n^{3/2})$.

Beyond the complete final normal form, we conjecture that the other constructions for possibly incomplete automata and automata with a fixed number of missing transitions could be adapted as well, and even retain the same complexities.

4.3. Experimentation: From DTWAs to Top-Down Tree Automata

Tree walking automata enjoy a tight connection with several logical formalisms [7, 9], including some XPath fragments. Formula satisfiability then reduces to the emptiness of the language of a tree walking automaton. Nevertheless, the latter problem is rather hard to decide: it is an ExpTime-complete problem, for which the known algorithms consist essentially in constructing an exponentially larger equivalent top-down tree automaton, and (on the fly) checking this automaton for emptiness in polynomial time.

We have implemented a prototype tool for converting DTWAs into coaccessible nondeterministic top-down tree automata (under the form of Relax NG grammars [6]). Given a DTWA with n states, the resulting top-down tree automaton can hold as many as $O(2^{n^2})$ states, that encode which pairs (p, q) of states allow a run of the DTWA to start from state p on a given tree node and return to it in state q without ever visiting its parent node.

We ran the algorithm on 100 randomly generated incomplete DTWA for each n and report the mean number of states in the computed equivalent top-down tree automaton in Figure 3. Due to very high standard deviation values, we exclude the 10 smallest and 10 largest output automata from the mean computation, and display their mean number of states on separate plots. All three plots display an exponential behaviour. Overall, the translation results in a $O(2^n)$ size increase on average, which is significantly better than the worst-case $O(2^{n^2})$ bound.

5. Application to Top-Down Tree Automata

5.1. Deterministic Top-Down Tree Automata

In this section, $\mathcal{F}$ denotes a finite ranked alphabet, i.e. there is an arity function ar from $\mathcal{F}$ into $\mathbb{N}$. We denote by $\mathcal{F}_i$ the subset of elements C of $\mathcal{F}$ such that $\text{ar}(C) = i$. We assume that

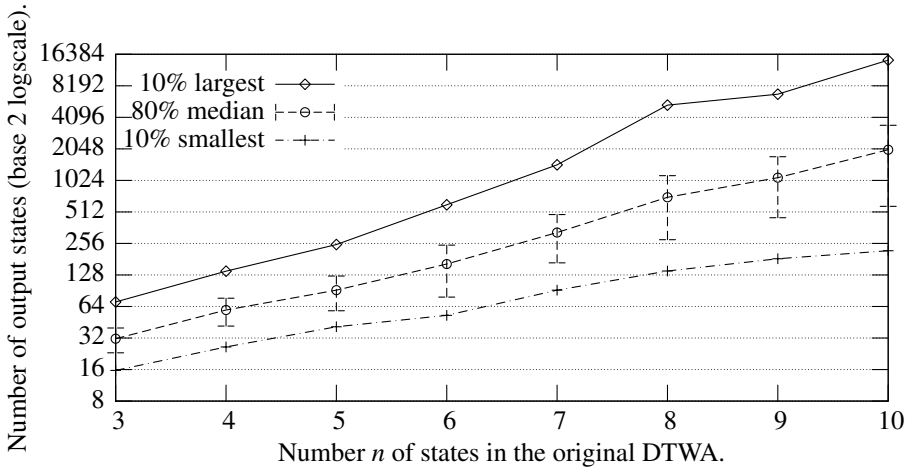


Figure 3: Average number of states in the 10 smallest, the 10 largest, and the 80 median top-down tree automata obtained from transforming 100 2-letter DTWAs with n states.

$\$ \notin \mathcal{F}$. Let $\overline{\mathcal{F}} = \{(f,i) \mid f \in \mathcal{F} \setminus \mathcal{F}_0,\ 1 \leq i \leq \mathrm{ar}(f)\}$.

A *deterministic top-down tree automata* (DTDA) is a tuple $(Q, \mathcal{F}, \theta, q_{\mathrm{init}})$ where Q is a finite set of *states* satisfying $0 \notin Q$, $q_{\mathrm{init}} \in Q$ is the *initial state*, and θ is a partial *transition function* mapping elements of $Q \times \mathcal{F}_i$ to Q^i (for all $i \geq 1$) and elements of $Q \times \mathcal{F}_0$ to 0. One can inductively define accessible states of a DTDA by: the initial state q_{init} is accessible and for every $f \notin \mathcal{F}_0$, if q is accessible and $\theta(q,f) = (q_1, \ldots, q_{\mathrm{ar}(f)})$ then the q_i's are accessible. A DTDA is *complete* if $Q \times (\mathcal{F} \setminus \mathcal{F}_0) \subseteq \mathrm{Dom}(\theta)$. For more information on top-down tree automata, the reader is referred to [19].

Let $\mathcal{A}_1 = (Q_1, \mathcal{F}, \theta_1, q_{\mathrm{init1}})$ and $\mathcal{A}_2 = (Q_2, \mathcal{F}, \theta_2, q_{\mathrm{init2}})$ be two DTDAs. An *isomorphism* φ is a bijective function φ from Q_1 to Q_2 such that (1) for every state q, every $f \in \mathcal{F} \setminus \mathcal{F}_0$, $\theta_1(q,f) = (q_1, \ldots, q_{\mathrm{ar}(f)})$ iff $\theta_2(\varphi(q), f) = (\varphi(q_1), \ldots, \varphi(q_{\mathrm{ar}(f)}))$, (2) $\varphi(q_{\mathrm{init1}}) = q_{\mathrm{init2}}$, and (3) for every state q, every $C \in \mathcal{F}_0$, $\theta_1(q,C) = 0$ iff $\theta_2(\varphi(q), C) = 0$.

5.2. From SLTs to DTDAs

We define in this section a bijection ψ from DTDAs to a subclass of SLTs, called *DTDA-coherent* SLTs, that contains all the complete SLTs. For every DTDA $\mathcal{A} = (Q, \mathcal{F}, \theta, q_{\mathrm{init}})$, let $\psi(\mathcal{A})$ be the SLT

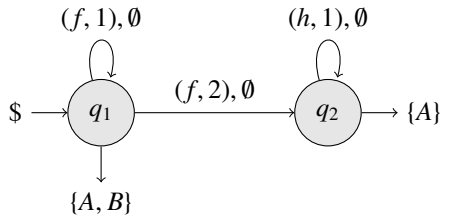
$$\psi(\mathcal{A}) = (\overline{\mathcal{F}}, \mathcal{P}(\mathcal{F}_0) \uplus \{\$\}, Q, q_{\mathrm{init}}, \delta, \gamma, \rho, \$)$$

defined by: $\gamma(q,(f,i)) = \emptyset$ and $\delta(q,(f,i)) = p_i$ iff $\theta(q,f) = (p_1, \ldots, p_n)$, and $\rho(q) = \{A \in \mathcal{F}_0 \mid \theta(q,A) = 0\}$ iff this set is not empty, and $\rho(q)$ is undefined otherwise. The mapping ψ is closely related to the path closure characterization of deterministic top-down tree languages [19].

For example, let $\mathcal{F}_0 = \{A, B\}, \mathcal{F}_1 = \{h\}$ and $\mathcal{F}_2 = \{f\}$ in the DTDA $\mathcal{A}_{\mathrm{ex}} = (\{q_1, q_2\}, \mathcal{F}, \theta_{\mathrm{ex}}, \{q_1\})$ with $\theta_{\mathrm{ex}}(q_1, f) = (q_1, q_2)$, $\theta_{\mathrm{ex}}(q_2, h) = q_2$, and $\theta_{\mathrm{ex}}(q_1, A) = \theta_{\mathrm{ex}}(q_1, B) = \theta_{\mathrm{ex}}(q_2, A) = 0$. This entails $\overline{\mathcal{F}} = \{(h,1), (f,1), (f,2)\}$ in the SLT $\psi(\mathcal{A}_{\mathrm{ex}})$ depicted in Figure 4.

A SLT $(\overline{\mathcal{F}}, \mathcal{P}^*(\mathcal{F}_0) \uplus \{\$\}, Q, q_{\mathrm{init}}, \delta, \gamma, \rho, \$)$ is *DTDA-coherent* if

1. for every state q, every $(f,i) \in \overline{\mathcal{F}}$, $\delta(q,(f,i))$ is defined iff $\delta(q,(f,j))$ is defined for all $j \in [1, ar(f)]$,
2. $\gamma(q,(f,i))$ is either undefined or equal to $\emptyset$, and



$$\delta_{\text{ex}}(q_1, (f, 1)) = q_1$$
$$\delta_{\text{ex}}(q_1, (f, 2)) = q_2$$
$$\delta_{\text{ex}}(q_2, (h, 1)) = q_2$$
$$\rho_{\text{ex}}(q_1) = \{A, B\}$$
$$\rho_{\text{ex}}(q_2) = \{A\}$$

Figure 4: The SLT $\psi(\mathcal{A}_{\text{ex}}) = (\overline{\mathcal{F}}, \mathcal{P}^*(\{A, B\}) \uplus \{\$\}, \{q_1, q_2\}, q_1, \delta_{\text{ex}}, \gamma_{\text{ex}}, \rho_{\text{ex}}, \$)$.

3. its initial output is $\$$.

Proposition 5. *The function ψ is a bijection from DTDA to DTDA-coherent SLTs. Moreover, for every DTDA $\mathcal{A}$, $\psi(\mathcal{A})$ is complete (resp. accessible) if and only if $\mathcal{A}$ is complete (resp. accessible).*

Proof. If $\mathcal{A}$ is a DTDA, then it is clear that $\psi(\mathcal{A})$ is DTDA-coherent. Let $\mathcal{A}_1 = (Q_1, \mathcal{F}, \theta_1, q_{\text{init1}})$ and $\mathcal{A}_2 = (Q_2, \mathcal{F}, \theta_2, q_{\text{init2}})$ be DTDAs such that $\psi(\mathcal{A}_1) = \psi(\mathcal{A}_2)$. By definition of ψ, $Q_1 = Q_2$ and $q_{\text{init1}} = q_{\text{init2}}$. Set $\psi(\mathcal{A}_1) = \psi(\mathcal{A}_2) = (\overline{F}, \mathcal{P}(\mathcal{F}_0) \uplus \{\$\}, Q_1, q_{\text{init1}}, \delta, \gamma, \rho, \$)$. Reasoning on δ shows that θ_1 and θ_2 are equal for letters in $\mathcal{F} \setminus \mathcal{F}_0$. Reasoning on ρ shows that θ_1 and θ_2 are equal for letters in $\mathcal{F}_0$. It follows that ψ is injective. The remaining points of the proposition are straightforward verifications. □

Proposition 6. *Two DTDAs $\mathcal{A}_1$ and $\mathcal{A}_2$ are isomorphic if and only if $\psi(\mathcal{A}_1)$ and $\psi(\mathcal{A}_2)$ are isomorphic.*

Proof. It suffices to note that the same isomorphism holds between $\mathcal{A}_1$ and $\mathcal{A}_2$ and $\psi(\mathcal{A}_1)$ and $\psi(\mathcal{A}_2)$. □

Let $r^{\text{DTDA}} = \overline{\mathcal{F}} \times \{\emptyset\}$. The class E_n^{DTDA} of DTDA-coherent SLTs with n states contains $C_n(\overline{\mathcal{F}}, \mathcal{P}(\mathcal{F}_0) \uplus \{\$\}, r^{\text{DTDA}}, \{\$\}, \mathcal{P}^*(\mathcal{F}_0))$ and is included in $\mathcal{D}_n(\overline{\mathcal{F}}, \mathcal{P}(\mathcal{F}_0) \uplus \{\$\}, r^{\text{DTDA}}, \{\$\}, \mathcal{P}^*(\mathcal{F}_0))$. Thus, random generation of DTDAs can be performed using Proposition 1 or Proposition 2 to obtain a SLT $\mathcal{T}$ and by computing $\psi^{-1}(\mathcal{T})$.

6. Beyond Tree Automata

We present in this section how to tailor our approach for the random generation of deterministic Turing Machines (Section 6.1), normalized deterministic pushdown automata (Section 6.2), and deterministic visibly pushdown automata (Section 6.3). These examples provide further testimony on the ease of adapting our uniform random generator for SLTs.

6.1. Deterministic Turing Machines

A *deterministic Turing machine* (DTM) is a tuple $(Q, \Sigma, \Delta, q_{\text{init}}, F)$ where Q is a finite set of *states*, Σ is a finite alphabet, $q_{\text{init}} \in Q$ is the *initial state*, $F \subseteq Q$ is the set of *final states* and Δ is a partial *transition function* from $Q \times \Sigma$ into $\Sigma \times Q \times \{\leftarrow, \rightarrow\}$. A deterministic Turing machine is *complete* if Δ is a function. Weakly-accessible states of a DTM are defined inductively: q_{init} is weakly-accessible, and if q is weakly-accessible and $\Delta(q, a) = (b, p, t)$ for some $a, b \in \Sigma$ and $t \in \{\leftarrow, \rightarrow\}$, then p is weakly-accessible. A DTM is *weakly-accessible* if all its states are weakly-accessible.

An *isomorphism* from a DTM $(Q_1, \Sigma, \Delta_1, q_{\text{init1}}, F_1)$ to a DTM $(Q_2, \Sigma, \Delta_2, q_{\text{init2}}, F_2)$ is a bijective function from Q_1 to Q_2 satisfying the three conditions (1) $\varphi(q_{\text{init1}}) = q_{\text{init2}}$, (2) $\varphi(q) \in F_2$ iff $q \in F_1$, and (3) $\Delta_1(q, a) = (b, p, t)$ iff $\Delta_2(\varphi(q), a) = (b, \varphi(p), t)$.

We define now a bijection ν between DTMs and a class of STLs, called *DTM-coherent* STLs, that contains all the complete STLs. As in previous cases, we obtain this way a random generation algorithm for DTMs.

Let $\mathcal{M} = (Q, \Sigma, \Delta, q_0, F)$ be a deterministic Turing machine. The SLT $\nu(\mathcal{M})$ is defined by

$$\nu(\mathcal{M}) = (\Sigma, \Sigma_2, Q, q_{\text{init}}, \delta, \gamma, \rho, \$) \ ,$$

with $\Sigma_2 = \Sigma \times (\{\leftarrow, \rightarrow\}) \uplus \{\$, 1\}$, $\delta(q, a) = p$ and $\gamma(q, a) = (b, t)$ iff $\Delta(q, a) = (b, p, t)$ $(t \in \{\leftarrow, \rightarrow\})$, and $\text{Dom}(\rho) = F$ with $\rho(q) = 1$ iff $q \in F$.

An SLT on Σ, $\Sigma \times (\{\leftarrow, \rightarrow\}) \uplus \{\$, 1\}$ is *DTM-coherent* if its initial output symbol is \$ and if for every $q \in \text{Dom}(\rho)$, $\rho(q) = 1$.

The two following propositions hold. Proofs are similar to the ones of Propositions 3 and 4 and are left to the reader.

Proposition 7. *The function ν is a bijection from DTMs to DTM-coherent SLTs. Moreover, for every DTM $\mathcal{M}$, $\nu(\mathcal{M})$ is complete (resp. accessible) if and only if $\mathcal{M}$ is complete (resp. weakly-accessible).*

Proposition 8. *Two DTMs $\mathcal{M}_1$ and $\mathcal{M}_2$ are isomorphic if and only if $\nu(\mathcal{M}_1)$ and $\nu(\mathcal{M}_2)$ are isomorphic.*

6.2. Normalized Real-time Deterministic Pushdown Automata

A *normalized real-time deterministic pushdown automaton* (NRDPDA) is a tuple of form $(Q, \Sigma, \Gamma, \Delta, q_{\text{init}}, Z_{\text{init}}, F)$ where Q is a finite set of *states*, Σ a finite alphabet, Γ a (finite) stack alphabet, $q_{\text{init}} \in Q$ the initial state, $Z_{\text{init}} \in \Gamma$ a distinguished symbol that serves as initial stack content, $F \subseteq Q$ the set of *final states*, and Δ a partial function from $Q \times \Gamma \times \Sigma$ into $Q \times \Gamma^*$ satisfying for any state q, letter $q \in \Sigma$ and stack symbol $X \in \Gamma$, that if $\Delta(q, X, a)$ is defined, then it is either of a *pop* transition to (q', ε), an *internal* transition to (q', X), or a *push* transition (q', XY) where $Y \in \Gamma$ and $q' \in Q$. A NRDPDA is complete if Δ is a total function.

Weakly-accessible states of a NRDPDA are defined inductively: q_{init} is weakly-accessible, and if q is weakly-accessible and $\Delta(q, X, a) = p$ for some $a \in \Sigma$ and $X \in \Gamma$, then p is weakly-accessible. An example of a NRDPDA is depicted in Figure 5.

An *isomorphism* from a NRDPDA $\mathcal{A}_1 = (Q_1, \Sigma, \Gamma, \Delta_1, q_{\text{init1}}, Z_{\text{init}}, F_1)$ to a NRDPDA $\mathcal{A}_2 = (Q_2, \Sigma, \Gamma, \Delta_2, q_{\text{init2}}, Z_{\text{init}}, F_2)$ is a bijective function form Q_1 to Q_2 satisfying (1) $\varphi(q_{\text{init1}}) = q_{\text{init2}}$, (2) $\varphi(q) \in F_2$ iff $q \in F_1$, and (3) $\Delta_1(q, a, X) = (p, t)$ iff $\Delta_2(\varphi(q), a, X) = (\varphi(p), t)$.

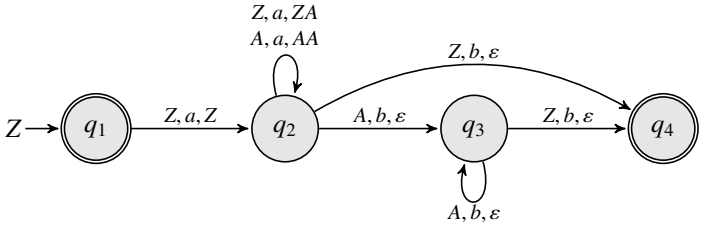


Figure 5: A normalized real-time deterministic pushdown automaton.

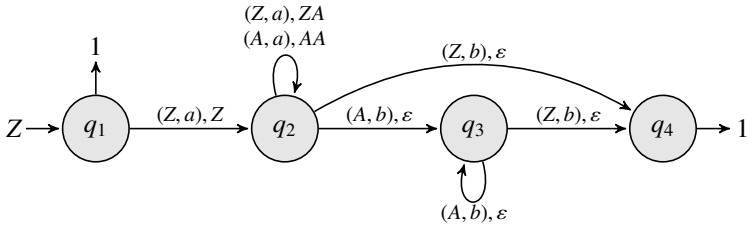


Figure 6: The NRDPDA-coherent SLT corresponding to the automaton of Figure 5.

Now we define a bijection η between normalized real-time deterministic pushdown automata and a class of SLTs, called *NRDPDA-coherent SLTs* that contains all complete SLTs. For every NRDPDA $\mathcal{A} = (Q, \Sigma, \Gamma, \Delta, q_{\text{init}}, F)$, the SLT $\eta(\mathcal{A})$ is defined by

$$\eta(\mathcal{A}) = (\Gamma \times \Sigma, \{\varepsilon\} \uplus \Gamma \uplus \Gamma^2 \uplus \{1\}, Q, q_{\text{init}}, \delta, \rho, Z_{\text{init}})$$

where $\delta(q, (X, a)) = p$ and $\rho(q, (X, a)) = \gamma$ iff $\Delta(q, X, a) = (p, \gamma)$, and $\text{Dom}(\rho) = F$ with $\rho(q) = 1$ iff $q \in F$. The image by η of the NRDPDA depicted in Figure 5 is shown in Figure 6.

An SLT on $\Gamma \times \Sigma$,$\{\varepsilon\} \uplus \Gamma \uplus \Gamma^2 \uplus \{1\}$ is *NRDPDA-coherent* if it fulfills the following conditions:

- its initial output symbol is Z_{init},
- for every $q \in \text{Dom}(\rho)$, $\rho(q) = 1$, and
- if $\delta(q, (X, a))$ is defined, then $\rho(q, (X, a)) \in \{\varepsilon\} \cup \{X\} \cup (\{X\} \times \Gamma)$.

Proposition 9. *The function η is a bijection from NRDPDAs to NRDPDA-coherent SLTs. Moreover, for every NRDPDA $\mathcal{A}$, $\eta(\mathcal{A})$ is complete (resp. accessible) if and only if $\mathcal{A}$ is complete (resp. weakly-accessible).*

Proof. The proposition is a direct consequence of the definition of η. □

Proposition 10. *Two NRDPDAs $\mathcal{A}_1$ and $\mathcal{A}_2$ are isomorphic if and only if $\eta(\mathcal{A}_1)$ and $\eta(\mathcal{A}_2)$ are isomorphic.*

Proof. It suffices to check that the same isomorphism holds between $\mathcal{A}_1$ and $\mathcal{A}_2$ and $\eta(\mathcal{A}_1)$ and $\eta(\mathcal{A}_2)$. □

Let $\Sigma_1 = \Gamma \times \Sigma$ and $\Sigma_2 = \{\varepsilon\} \uplus \Gamma \uplus \Gamma^2 \uplus \{1\}$. For each a in Σ and X in Γ, let $r^{\mathrm{PDA}}(X, a) = \{\varepsilon\} \cup \{X\} \cup (\{X\} \times \Gamma)$. The class E_n^{PDA} of NRDPDA-coherent SLTs with n states contains $C_n(\Sigma_1, \Sigma_2, r^{\mathrm{PDA}}, \{Z_{\mathrm{init}}\}, \{1\})$ and is included in $\mathcal{D}_n(\Sigma_1, \Sigma_2, r^{\mathrm{PDA}}, \{Z_{\mathrm{init}}\}, \{1\})$. Thus, random generation of NRDPDAs can be performed using Proposition 1 or Proposition 2 to obtain a SLT $\mathcal{T}$ and by computing $\eta^{-1}(\mathcal{T})$.

6.3. Deterministic Visibly Pushdown Automata

Deterministic visibly pushdown automata (DVPA) form a subclass of the normalized real-time deterministic pushdown automata of Section 6.2. Visibly pushdown automata were introduced by Alur and Madhusudan [12] as a robust class of context-free languages fit for program analysis and tree language representation. In particular, DVPAs can represent all regular tree languages—which is neither the case of DTWAs nor DTDAs—including languages of unbounded trees, i.e. where the arity of symbols in not bounded.

A DVPA operates on an input alphabet Σ divided into three disjoint subsets Σ_c of *calls*, Σ_i of *internal* actions, and Σ_r of *returns*. The automaton itself is a tuple $\mathcal{A} = (Q, \Sigma, \Gamma, \Delta, q_{\mathrm{init}}, F)$ where Γ contains a distinguished bottom-of-stack symbol $\perp$, and Δ is the union of three functions $\Delta_c : Q \times \Sigma_c \to (\Gamma \backslash \{\perp\}) \times Q$ for push transitions, $\Delta_r : Q \times \Sigma_r \times \Gamma \to Q$ for pop transitions, and $\Delta_i : Q \times \Sigma_i \to Q$ for internal transitions: the input symbol constrains the type of transition that the automaton can make. An example of a DVPA equivalent to the NRDPDA of Figure 5 is shown in Figure 7.

The translation to SLTs is very similar to that of NRDPDAs, and we merely sketch it: define $\mu(\mathcal{A})$ as the SLT

$$\mu(\mathcal{A}) = (\Sigma_c \cup (\Sigma_r \times \Gamma) \cup \Sigma_i, (\Gamma \backslash \{\perp\}) \uplus \{\$, 1\}, Q, q_{\mathrm{init}}, \delta, \rho, \$)$$

where

- the initial output is always \$, thanks to the restriction $r_i^{\mathrm{VPA}} = \{\$\}$,
- for $a \in \Sigma_c$ and $Z \in \Gamma \backslash \{\perp\}$, $\delta(q, a) = p$ and $\rho(q, a) = Z$ iff $\Delta_c(q, a) = (p, Z)$, which is obtained through the restriction $\forall a \in \Sigma_c, r^{\mathrm{VPA}}(a) = \Gamma \backslash \{\perp\}$,
- for $b \in \Sigma_r$ and $Z \in \Gamma$, $\delta(q, (b, Z)) = p$ and $\rho(q, (b, Z)) = 1$ iff $\Delta_r(q, b, Z) = p$, which is obtained through the restriction $\forall b \in \Sigma_r, \forall Z \in \Gamma, r^{\mathrm{VPA}}((b, Z)) = \{1\}$,
- for $c \in \Sigma_i$, $\delta(q, c) = p$ and $\rho(q, c) = 1$ iff $\Delta_i(q, c) = p$, which is obtained through the restriction $\forall c \in \Sigma_i, r^{\mathrm{VPA}}(c) = \{1\}$, and
- $\mathrm{Dom}(\rho) = F$ with $\rho(q) = 1$ iff $q \in F$, which is obtained through the restriction $r_F^{\mathrm{VPA}} = \{1\}$.

As always, using Proposition 1 or Proposition 2 and the above restrictions, we obtain a suitable SLT $\mathcal{T}$ from which the desired DVPA can be computed as $\mu^{-1}(\mathcal{T})$. Note however that the complexity bounds we have disappear if Γ grows too large, i.e. becomes commensurate with n, which is the case in the translation of Alur and Madhusudan [12] from regular tree languages to visibly pushdown languages. Furthermore, unlike the tree languages of the DTDAs that we generated in Section 5, the languages of our DVPAs might be empty since we can only guarantee weak accessibility.

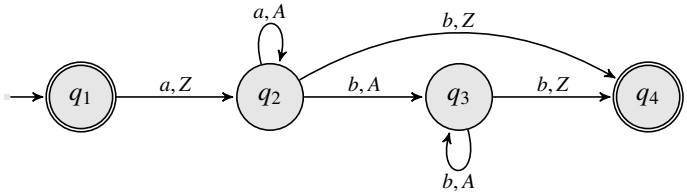


Figure 7: A deterministic visibly pushdown automaton with $\Gamma = \{\bot, Z, A\}$, $\Sigma_c = \{a\}$, $\Sigma_i = \emptyset$, and $\Sigma_r = \{b\}$.

7. Conclusion

In this paper we define a rejection algorithm to randomly and uniformly generate sequential letter-to-letter transducers parametrized with output restrictions and/or a fixed number of transitions. We exhibit two bijections from this class of transducers to the class of deterministic tree walking automata and deterministic top-down tree automata respectively, and report on an empirical evaluation of a $O(2^n)$ average complexity instead of a $O(2^{n^2})$ worst-case bound for turning a deterministic tree walking automaton into an equivalent nondeterministic top-down tree automaton.

We show that the approach we propose in this paper is straightforward to use on other classes of finite state machines, like deterministic Turing machines or some classes of pushdown automata. By tailoring the restrictions, we can even generate deterministic visibly pushdown automata, which recognize (encodings) of all regular tree languages. This is still somewhat unsatisfactory from a tree language viewpoint, but a much less obvious variation would be needed in order to randomly generate deterministic bottom-up tree automata or hedge automata.

References

[1] M. D. Wulf, L. Doyen, T. A. Henzinger, J.-F. Raskin, Antichains: A new algorithm for checking universality of finite automata, in: T. Ball, R. B. Jones (Eds.), CAV'06, volume 4144 of *Lecture Notes in Computer Science*, Springer, 2006, pp. 17–30.

[2] R. J. van Glabbeek, B. Ploeger, Five determinisation algorithms, in: O. H. Ibarra, B. Ravikumar (Eds.), CIAA'08, volume 5148 of *Lecture Notes in Computer Science*, Springer, 2008, pp. 161–170.

[3] S. Schewe, Büchi complementation made tight, in: S. Albers, J.-Y. Marion (Eds.), STACS'09, volume 3 of *Leibniz International Proceedings in Informatics*, Schloss Dagstuhl - LCI, 2009, pp. 661–672.

[4] F. Bassino, J. David, C. Nicaud, On the average complexity of Moore's state minimization algorithm, in: S. Albers, J.-Y. Marion (Eds.), STACS'09, volume 3 of *Leibniz International Proceedings in Informatics*, Schloss Dagstuhl - LCI, 2009, pp. 123–134.

[5] F. Neven, Automata theory for XML researchers, SIGMOD Record 31 (2002) 39–46.

[6] M. Murata, D. Lee, M. Mani, K. Kawaguchi, Taxonomy of XML schema languages using formal language theory, ACM Transactions on Internet Technology 5 (2005) 660–704.

[7] J. Engelfriet, H. J. Hoogeboom, Tree-walking pebble automata, in: J. Karhumäki, H. A. Maurer, G. Paun, G. Rozenberg (Eds.), Jewels are Forever, Springer, 1999, pp. 72–83.

[8] M. Bojańczyk, T. Colcombet, Tree-walking automata do not recognize all regular languages, SIAM J. Comput. 38 (2008) 658–701.

[9] B. ten Cate, L. Segoufin, XPath, transitive closure logic, and nested tree walking automata, in: M. Lenzerini, D. Lembo (Eds.), PODS'08, ACM, 2008, pp. 251–260.

[10] D. Tabakov, M. Y. Vardi, Experimental evaluation of classical automata constructions, in: G. Sutcliffe, A. Voronkov (Eds.), LPAR'05, volume 3835 of *Lecture Notes in Computer Science*, Springer, 2005, pp. 396–411.

[11] A. Bouajjani, P. Habermehl, L. Holík, T. Touili, T. Vojnar, Antichain-based universality and inclusion testing over nondeterministic finite tree automata, in: O. H. Ibarra, B. Ravikumar (Eds.), CIAA'08, volume 5148 of *Lecture Notes in Computer Science*, Springer, 2008, pp. 57–67.

[12] R. Alur, P. Madhusudan, Visibly pushdown languages, in: STOC'04, ACM, 2004, pp. 202–211.
[13] F. Bassino, C. Nicaud, Enumeration and random generation of accessible automata, Theor. Comput. Sci. 381 (2007) 86–104.
[14] F. Bassino, J. David, C. Nicaud, Enumeration and random generation of possibly incomplete deterministic automata, Pure Mathematics and Applications 19 (2008) 1–16.
[15] J.-M. Champarnaud, T. Paranthoën, Random generation of DFAs, Theor. Comput. Sci. 330 (2005) 221–235.
[16] F. Bassino, J. David, C. Nicaud, `REGAL`: A library to randomly and exhaustively generate automata, in: J. Holub, J. Žďárek (Eds.), CIAA'07, volume 4783 of *Lecture Notes in Computer Science*, pp. 303–305.
[17] Y.-F. Chen, A. Farzan, E. M. Clarke, Y.-K. Tsay, B.-Y. Wang, Learning minimal separating DFA's for compositional verification, in: S. Kowalewski, A. Philippou (Eds.), TACAS'09, volume 5505 of *Lecture Notes in Computer Science*, Springer, 2009, pp. 31–45.
[18] A. Denise, P. Zimmermann, Uniform random generation of decomposable structures using floating-point arithmetic, Theor. Comput. Sci. 218 (1999) 233–248.
[19] H. Comon, M. Dauchet, R. Gilleron, C. Löding, F. Jacquemard, D. Lugiez, S. Tison, M. Tommasi, Tree Automata Techniques and Applications, 2007.